

Имплементација подршке MIPS32 архитектуре у Dart SDK и Flutter окружењу

Implementation of MIPS32 architecture support in Dart SDK and Flutter framework

Славиша Петровић, Факултет техничких наука, Нови Сад

Студијски програм – РАЧУНАРСТВО И АУТОМАТИКА

Кратак садржај - У овом раду описана је имплементација подршке MIPS32 процесорске архитектуре у Dart SDK и Flutter окружењу. Теоријски део обухвата архитектуру Dart виртуелне машине, моделе извршавања (JIT и AOT), управљање меморијом, као и MIPS32 архитектуру, њен инструкцијски скуп и бинарни интерфејс апликације (ABI). Имплементациони део описује све кључне компоненте позадинског слоја преводиоца: асемблер за MIPS32 инструкцијски скуп, распоред стек оквира, позадински слој међурепрезентације са имплементацијама аритметичких и управљачких операција, координатор преводиоца и потпрограми извршног система. Поред Dart VM позадинског дела, описане су и измене у систему за изградњу Flutter engine-a.

Кључне речи: Dart SDK, Flutter, Flutter engine, компајлер, MIPS32

Abstract – This paper describes the implementation of MIPS32 processor architecture support in the Dart SDK and Flutter framework. The theoretical section covers the Dart virtual machine architecture, execution models (JIT and AOT), memory management, as well as the MIPS32 architecture, its instruction set, and application binary interface (ABI). The implementation section describes all key components of the compiler backend: the assembler for the MIPS32 instruction set, stack frame layout, the IL backend with implementations of arithmetic and control flow operations, the compilation coordinator, and runtime stubs. In addition to the Dart VM backend, the paper also covers the changes made to the Flutter engine build system.

Keywords: Dart SDK, Flutter, Flutter engine, compiler, MIPS32

НАПОМЕНА: Овај рад проистекао је из мастер рада чији ментор је био проф. др Иван Каштелан.

1. УВОД

Експанзија уређаја заснованих на различитим процесорским архитектурама, од мобилних уређаја и таблета до уграђених система у индустријским постројењима и потрошачкој електроници, довела је до

ситуације у којој је подршка за више платформи постала незаобилазна. У том контексту, Flutter се наметнуо као један од водећих вишеплатформских развојних оквира. Развијен од компаније Google, Flutter омогућава писање апликација које се потом могу покренути на различитим оперативним системима као што су Android, iOS, Linux, macOS и Windows, захваљујући сопственом механизму рендеровања који не зависи од уграђених графичких компоненти оперативног система [1].

Програмски језик који стоји у основи Flutter екосистема је Dart, статички типизиран, објектно оријентисан језик, такође развијен од стране компаније Google. Dart се преводи директно у машински код за циљну платформу, што осигурава задовољавајући ниво перформанси како у фази развоја (JIT), тако и у продукцији (AOT). Упркос овим предностима, екосистем Flutter-a и Dart-a карактерисала је једна значајна празнина: одсуство подршке за MIPS32 процесорску архитектуру [2, 3].

MIPS32 је архитектура редукованог скупа инструкција (RISC) присутна деценијама у уграђеним системима широм света. Милиони уређаја заснованих на MIPS32 процесорима, као што су кабловски и сателитски телевизијски пријемници, мрежни рутери и индустријски контролери, и данас су активно у употреби. Dart SDK је поседовао основну подршку за MIPS32 све до 2017. године, када је компанија Google одлучила да обустави њено даље одржавање. Последица је била искљученост читавог екосистема модерних Flutter апликација за MIPS платформу.

Циљ овог рада је да премости ту празнину, да MIPS32 архитектура поново буде равноправна циљна платформа у Dart SDK, и да се тиме отвори пут ка покретању Flutter апликација на уређајима који су до сада били искључени из тог екосистема. Поред академског доприноса, рад има и непосредну практичну вредност за компаније које развијају и одржавају софтвер за MIPS32 уређаје [3].

2. ТЕОРИЈСКЕ ОСНОВЕ

2.1. Dart виртуелна машина и модели извршавања

Dart VM (Virtual Machine) није виртуелна машина у класичном смислу речи. Dart код се не извршава тумачењем међукода као у Javi, него се преводи у

машински код и извршава директно на процесору. Термин “виртуелна машина” овде означава скуп компоненти које заједно омогућавају извршавање Dart програма: компајлере, управљање меморијом и систем за подршку током рада програма [2, 4].

Dart VM подржава два основна модела извршавања. JIT (Just-in-Time) превод се користи током развоја апликације и омогућава превод кода у тренутку извршавања, што допушта брзо тестирање промена без поновног покретања апликације (hot reload). AOT (Ahead-of-Time) превод се користи у продукцијским окружењима где се цео програм унапред преводи у машински код, што доноси брже покретање и боље перформансе. Кључна компонента АОТ компајлера је анализа тока типова (TFA), која прегледа цео програм и утврђује које делове кода програм заиста може да позове, што компајлеру омогућава да уклони непотребни код [4, 5].

Dart VM имплементира генерацијски сакупљач смећа који се заснива на хипотези да већина објеката кратко живи. Heap је подељен на млађу генерацију за новоалоциране објекте и стару генерацију за објекте дугог животног века. VM користи и технику “write barriers” за праћење референци из старе у младу генерацију, што омогућава ефикасно парцијално скупљање без скенирања целог heap-а.

2.2. Flutter архитектура и embedder

Flutter је дизајниран као проширив, слојевити систем. Централна апстракција су widget-и, непроменљиви описи делова корисничког интерфејса, који се комбинују у стабло. Flutter користи три међусобно усклађена стабла: widget tree, element tree, render tree. Сваки фрејм који Flutter прикаже пролази кроз дефинисани “rendering pipeline” који обухвата фазе изградње, распореда, цртања, композиције и растеризације.

Flutter Engine је срце система, писан претежно у C++ програмском језику, који обезбеђује све услуге ниског нивоа: графичко рендеровање (Skia/Impeller), текстуалну подршку, Dart извршни систем и управљање ресурсима. Engine комуницира са оперативним системом кроз тзв. embedder слој, платформски специфичну компоненту која имплементира јавни Flutter Embedder API. Управо ова архитектура омогућава портовање Flutter-а на нову платформу имплементацијом релативно танког embedder слоја, без потребе за модификовањем самог engine кода или Flutter окружења. За уграђене Linux системе користи се пројекат flutter-embedded-linux, који има минималну зависност од системских библиотеки и подржава више позадинских слојева за приказ [1, 6].

2.3. MIPS32 процесорска архитектура

MIPS32 је архитектура редукованог скупа инструкција (RISC) са 32 регистра опште намене ширине 32 бита. Све инструкције имају фиксну дужину од 32 бита и сврставају се у један од три основна формата: R тип (регистарски), I тип (непосредни операнд) и J тип (скок). Архитектура користи O32 бинарни интерфејс апликације (ABI), по коме се аргументи прослеђују у

регистрима A0-A3, а вредности враћају у V0-V1. Регистри S0-S7 су callee-saved и морају бити очувани кроз позиве функција [7, 8].

Специфичност која директно утиче на имплементацију преводиоца је механизам одложеног извршавања (delay slot). Када процесор наиђе на инструкцију гранања, инструкција која се физички налази одмах после ње у меморији увек се извршава. Ова инструкција се назива “слот кашњења”. MIPS32 такође нема посебан регистар стања (Status Register) какав постоји на ARM или x86 архитектури. Поређења се врше експлицитним инструкцијама које резултат уписују у регистар опште намене [7].

Као хардверска платформа за верификацију коришћена је развојна плоча MIPS Creator CI20 (MIPS32r2 1,2 GHz, 1 GB RAM, Debian Linux за MIPS little-endian). Поред физичке плоче, у свакодневном развоју коришћен је QEMU емулатор који подржава MIPS32 (MIPS little-endian) [9, 10].

3. ИМПЛЕМЕНТАЦИЈА

Преношење компајлера на нову архитектуру није тривијална измена, него скуп међусобно зависних компоненти које заједно чине потпун позадински слој (backend). Dart VM компајлер је дизајниран са јасним раздвајањем архитектурно независног и зависног дела. Све што је специфично за циљну архитектуру смешта се у скуп датотека са суфиксом _mips. Имплементација је заснована на три ослонаца: делимични порт из 2017. године, референтна 32-битна ARM имплементација (која дели специфичности са MIPS32 у погледу адресног простора и конвенције позива), и код писан од нуле за специфичности MIPS ISA.

3.1. Регистарски модел и конвенције позива

Сваки део компајлерског позадинског слоја ради са симболичким именима регистра, а не са бројевима. MIPS32 архитектура има 32 целобројна регистра опште намене (R0-R31) и 32 регистра са покретним резом (F0-F31). Имплементација почиње дефинисањем ових скупова у датотеци constant_mips.h. Поред набројивог типа Register, овај фајл дефинише и скупове регистра изражене као 32-битне маске типа RegList, којима алокатор регистра зна које регистре сме слободно да употреби, које мора да сачува и које треба у потпуности да избегне.

Dart VM резервише одређене регистре за интерне потребе (Dart специфични алијаси). Одлуке о додели биле су условљене O32 ABI. Dart регистри морају бити сачувани кроз C позиве (callee-saved), јер виртуелна машина позива C++ извршни систем и очекује да им вредности остану нетакнуте по повратку. Тако је THR (показивач на текућу нит) додељен S3, PP (показивач на object pool) додељен S7, а CODE_REG (показивач на текући објекат кода) додељен S6.

MIPS32 не поседује посебан регистар стања какав постоји на ARM или x86. То значи да тип Condition не представља бит у хардверском регистру стања, него апстрактну вредност (enum) коју чвор међурепрезентације носи уз себе. Асемблер ту

вредност разрешава у конкретну секвенцу инструкција приликом емитовања.

3.2. Имплементација асемблера

Асемблер у контексту Dart VM није засебан алат, него C++ класа која излаже методе за емитовање MIPS инструкција директно у меморијски бафер. MIPS32 скуп инструкција дефинише три основна формата, па асемблер имплементира одговарајуће методе `EmitRType`, `EmitIType` и `EmitJType` које примају симболичке аргументе и уграђују их у 32-битну реч у складу са MIPS32 стандардом. Сваки позив методе `Emit` уписује тачно четири бајта у бафер и ажурира стање механизма одложеног извршавања.

Механизам одложеног извршавања (`delay slot`) директна је последица проточне архитектуре MIPS32 процесора. Асемблер излаже механизам `delay_slot()` којим позивалац може да одложени слот попуни корисном инструкцијом уместо `NOP` инструкцијом, чиме се избегава губитац процесорског циклуса при сваком гранању. Стање механизма прати се кроз два поља: `delay_slot_available` (поставља се одмах након слања гранајуће инструкције) и `in_delay_slot` (поставља се током уписа инструкције у одложени слот). Сваки позив `Emit` ресетује оба поља, спречавајући ненамерно уписивање.

Све MIPS инструкције I типа располажу само 16-битним пољем за непосредни операнд. Учитавање произвољне 32-битне вредности захтева две инструкције: `lui` (пуни горњих 16 бита) и `ori` (пуни доњих 16 бита). Метода `LoadImmediate` ову секвенцу оптимизује. Вредности које стају у 16 бита решава једном `addiu` инструкцијом, а када су доњих 16 бита нула, `ori` се изоставља. За дугачке скокове чији помак не стаје у 16-битно поље MIPS условних грана, користи се механизам “*far branches*”. Инвертована грана прескаче блок за безусловни скок, а циљна адреса се учитава у регистар T9 путем `lui/ori` пара. Избор T9 није случајан; O32 конвенција захтева да T9 увек садржи адресу одредишне функције.

3.3. Распоред стек оквира и координатор превођења

Свака Dart функција почиње успостављањем стек оквира. Метода `EnterDartFrame` чува четири речи на стеку (RA, FP, CODE_REG, PP), поставља нови FP и резервише простор за локалне променљиве. Распоред је дефинисан константама у `stack_frame_mips.h` које изражавају офсете у бајтовима релативно на FP регистар. Свака промена распореда у `EnterDartFrame` мора бити праћена одговарајућом променом константи, јер их сакупљач смећа и деоптимизатор читају независно један од другог.

Датотека `flow_graph_compiler_mips.cc` имплементира координатора који пролази кроз граф међурепрезентације и на свакој инструкцији позива `EmitNativeCode`.

За сваку тачку деоптимизације у оптимизованом коду, `CreateDeoptInfo` бележи комплетно виртуално стање оквира у формату који извршни систем користи да реконструише неоптимизовани оквир. При

деоптимизацији, извршни систем чита ову табелу и изграђује неоптимизовани стек оквир пре наставка извршавања кроз секвенцу извршних потпрограма (`runtime stubs`).

3.4. Позадински слој међурепрезентације

Датотека `il_mips.cc` имплементира архитектурно специфичан део преводиоца који сваки чвор међурепрезентације преводи у конкретне MIPS32 инструкције. Сваки чвор имплементира две методе: `MakeLocationSummary` (декларише захтеве чвора за регистрима у првом пролазу, пре алокације) и `EmitNativeCode` (емитује стварни машински код користећи регистре које је алокатор доделио у другом пролазу). Ово раздвајање је кључно за преносивост, алокатор регистара не зна ништа о MIPS32, већ ради са апстрактним захтевима.

MIPS32 R тип инструкције су тро-операндне: `addu rd, rs, rt` што значи да резултати иду у `rd` без измене улазних операнда. Ова особина је експлицитно искоришћена у `MakeLocationSummary` за `BinarySmiOpInstr`: излаз може бити трећи, независни регистар, што елиминише непотребне `mov` инструкције. Најсложенији случај је множење. MIPS32 нема инструкцију која чува 32-битни резултат у једном регистру опште намене. Инструкција `mult rs, rt` израчунава 64-битни производ и смешта га у специјални пар HI:LO, а једини начин приступа је кроз `mflo` и `mfhi` инструкције.

3.5. Имплементација у Flutter окружењу

Само присуство MIPS позадинског слоја у Dart VM није довољно да Flutter апликација може бити изграђена и покренута на MIPS уређају. Неопходно је и да Flutter окружење буде правилно конфигурирано за MIPS на нивоу система за изградњу. Flutter engine се гради као динамичка библиотека `libflutter_engine.so` која садржи и Dart VM преведен за циљну архитектуру. Кључна GN променљива `dart_target_arch` мора бити експлицитно постављена на “*mips*”, јер назив који GN користи за MIPS little-endian (`mipsel`) не одговара директно имену које Dart SDK користи.

Централна измена у скрипти за конфигурацију Flutter engine-а била је додавање `mipsel` као дозвољене вредности за аргумент `--linux-cpu`, као и експлицитно постављање `dart_target_arch` на “*mips*”. Без ове корекције, систем за изградњу не би пронашао исправне изворне фајлове као што су `il_mips.cc`. Измене су биле потребне и у `fml/build_config.h` као и `tonic/build_config.h` фајловима, који дефинишу макрое за детекцију циљне платформе. Без одговарајућег блока за MIPS детекцију, преводилац би прескочио MIPS грану и оставио кључне променљиве архитектуре недефинисаним, узрокујући грешке у току превођења.

4. ВЕРИФИКАЦИЈА И РЕЗУЛТАТИ

Dart SDK садржи свеобухватан скуп тестова у директоријуму `tests/` организованих по слојевима система. Скуп тестова обухвата неколико категорија: `language/` тестови (4724 фајла) проверавају

спецификацију Dart језика укључујући типове, контролу тока, асинхроне конструкције и генеричке типове; corelib/ тестови (око 440 фајлова) покривају стандардне библиотеке попут dart::core и dart::collection; ffi/ тестови (око 200 фајлова) верификују позивање C функција и прослеђивање структура, чиме директно проверавају исправност O32 ABI имплементације; standalone/ тестови (око 480 фајлова) проверавају изолате и управљање процесима; док vm/ тестови укључују C++ тестове интерних механизма попут асемблера, алокатора објеката и сакупљача смећа, као и Dart тестове напредних ЈТ сценарија попут деоптимизације. Тестови су покренути на скупу од преко 12.500 конфигурација.

Коначно стање имплементације, мерено на том скупу, приказано је у Табели 1. Као референца коришћена је ARM имплементација, Табела 2.

Табела 1. Резултати MIPS32 имплементације

	Пролаз	Пад	Успешност
Release	12536	130	98,97%
Debug	12522	150	98,82%

Табела 2. Резултати ARM32 референтне имплементације

	Пролаз	Пад	Успешност
Release	12623	107	99,16%
Debug	12605	111	99,13%

Разлика између MIPS32 release конфигурације (130 палих тестова) и ARM имплементације (107 палих тестова) износи свега 23 теста, што представља разлику мању од 0,2% у укупном скупу. Ниједан од преосталих падова не указује на грешку у генерисању кода за аритметику, контролу тока или управљање меморијом, што су критични предуслови за функционалан ЈТ преводилац.

Поред аутоматизованих тестова, верификација је спроведена и на физичкој MIPS Creator CI20 плочи. Више Flutter апликација покренуто је директно на хардверу: основна “hello_world” апликација која исписује текст, стандардна “flutter create” демо апликација, као и пример који приказује нативне текстуре унутар Flutter UI-а. Све приказане апликације покренуте су без модификација изворног кода.

5. ЗАКЉУЧАК

У овом раду описана је имплементација подршке MIPS32 процесорске архитектуре у Dart SDK и Flutter окружењу. Поред Dart VM, описане су и неопходне измене у Flutter engine систему за изградњу.

Посматрано из перспективе крајњег корисника, резултат је конкретан и одмах употребљив. Програмер који развија Flutter апликацију за модерни мобилни уређај може ту исту апликацију превести и покренути на MIPS платформи без измене иједне линије свог апликативног кода. Верификација на скупу од преко 12.500 тестова показује стопу успешности од 98,97% у release конфигурацији, што је мање од 0,2% испод референтне ARM имплементације. Успешно покретање више Flutter примера на физичкој MIPS

Creator CI20 плочи потврђује функционалан “end-to-end” ток од Dart изворног кода до пиксела на екрану.

Имплементација приказана у овом раду није замишљена као изолована грана, већ као основа за интеграцију у званичну Google грану.

Овим радом је показано да MIPS архитектура, упркос томе што је често занемарена у контексту савременог мобилног развоја, није техничка препрека за покретање Flutter апликација; препрека је била једино одсуство имплементације.

6. ЛИТЕРАТУРА

- [1] Flutter Team, Google. “Flutter Architectural Overview.” [Online]. Доступно: <https://docs.flutter.dev/resources/architectural-overview>. (приступљено у априлу 2026.).
- [2] V. Egorov. “Dart VM Internals.” [Online]. Доступно: <https://mrle.ph/dartvm>. (приступљено у априлу 2026.).
- [3] Dart Team, Google. “Dart SDK.” GitHub. [Online]. Доступно: <https://github.com/dart-lang/sdk>. (приступљено у априлу 2026.).
- [4] V. Egorov. “Introduction to Dart VM.” [Online]. Доступно: <https://plugfox.dev/introduction-to-dart-vm/>. (приступљено у априлу 2026.).
- [5] Dart Team, Google. “Dart Overview.” [Online]. Доступно: <https://dart.dev/overview>. (приступљено у априлу 2026.).
- [6] Sony Group Corporation. “flutter-embedded-linux.” GitHub. [Online]. Доступно: <https://github.com/sony/flutter-embedded-linux>. (приступљено у априлу 2026.).
- [7] MIPS Technologies, Inc. MIPS32 Architecture Reference Manual, Rev. 5.03. MIPS Technologies, Inc., 2016.
- [8] Linux Foundation. “System V ABI – MIPS RISC Processor Supplement,” 3rd ed. [Online]. Доступно: <https://refspecs.linuxfoundation.org/elf/mipsabi.pdf>. (приступљено у априлу 2026.).
- [9] eLinux.org. “CI20 Hardware Specifications.” [Online]. Доступно: https://elinux.org/CI20_Hardware. (приступљено у априлу 2026.).
- [10] eLinux.org. “MIPS Creator CI20.” [Online]. Доступно: https://elinux.org/MIPS_Creator_CI20. (приступљено у априлу 2026.).

Кратка биографија:

Славиша Петровић рођен је у Лозници 2001. године. Дипломске студије на Факултету техничких наука у Новом Саду из области Електротехнике и рачунарства завршио је 2024. године.

Контакт: petrovicslavisa96@gmail.com